NASA TT-20317

NASA TECHNICAL TRANSLATION

NASA TT - 20317

DIRECTIONS FOR PROGRAMMING IN FORTRAN 77

Hildegard Schuster

Translation of "RICHTLINIEN ZUR PROGRAMMIERUNG IN FORTRAN 77".
DFVLR-Mitt. 87-10, 1987, pp. 1-41

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D. C. 20546 JULY 1988

TT-20317

DFVLR Deutsche Forschungs- und Versuchsanstalt /1[*]
fuer Luft- und Raumfahrt

German Research and Test Institute for Air and Space Travel

Report

Directions for programming in FORTRAN 77

Hildegard Schuster

DFVLR
Main Central Data Processing Department, Oberpfaffenhofen

41 pages, 9 references

DFVLR-Mitt. 87-10

Manuscript submitted, July 2, 1987 /2

[*] Numbers in margin indicate foreign pagination.

Directions for programming in FORTRAN 77 /3

Version 1.0

Deutsche Forschungs- und Versuchsanstalt fuer Luft- und Raumfahrt
Field: Scientific and technical operational systems
Main division, central data processing
Integrated end systems division
D-8031 Oberpfaffenhofen, Post Wessling/Obb.

Oberpfaffenhofen. December 1986

Main division director:
Dr. rer. nat. H.-M. Wacker

Division director:
Dipl.-Ing. K. Weigand

Author:
Dipl.-Inf. (FH) H. Schuster

/4

FORTRAN 77, Programming Directions, Structured Programming

FORTRAN 77 Programming Directions

Summary

With the introduction of programming directions it is possible to reduce long-term costs of software development and maintenance.

The foundation of every programming should be based on the rules of the structured programming. For FORTRAN 77 programming these rules are insufficient, therefore further rules must be defined. Some of these programming rules can be simply used for other programming languages.

Table of contents

/6

1. Rules for programming - coding /7

1.1 General rules for setting up program

1. A program system should be constructed in modular form, i.e. always consisting of a control module (= main program and possibly one or more sub-programs) and one or more function modules (e.g. sub-programs). A program draft according to The EVA rule[1] (input, processing, output) meets this requirement. In addition, functions that occur more than once belong in sub-programs (algorithms, input/output functions).

   REASON - EXPLANATION

   A modular form of program construction significantly increases the quality of a program. Especially since all modules (= logical program unit / one or more procedures) can first be tested singly and then together in an integration test.
   Another advantage is that the effects of other changes are calculable as a result of isolation of functions in separate modules.

2. Each procedure (main or sub-program = sub-routine/function) should possess a clear input and output. Therefore the"ENTRY"and "RETURN i" instructions are forbidden in all procedures. The "STOP" instruction is permitted only once in the main program and in an error processing routine.

   REASON - EXPLANATION

   In the sense of structured programming, each block (see also p. 23) should possess a definite input and output. Programs that stick to structured programming are first of all easier to change /8 and secondly errors can be found more easily. Since the "ENTRY" instruction represents a further input and the "RETURN i" or "STOP" instruction represents a further output, these instruc-

[1] The EVA rule is explained in detail in many books on the subject of software engineering.

tions are not permissible.

3. Except in loops, the program flow should always be in a forwards direction.

   REASON - EXPLANATION

   According to the rules for structured programming, no jumps are allowed in the program. If parts of theprogram are not to be carried out, then the "IF - THEN - ELSE - ENDIF" structure should be used. Parts of the program that are used more than once, at different points, should be replaced by sub-programs.

4. Each variable should be explicitly declared.

   REASON - EXPLANATION

   All new compilers offer the possibility of listing all the parameters used in one procedure with their attributes. Using the reference list, all the names can be monitored and, for example, those that resulted from writing error can be corrected. In addition, especially helpful is a compiler that supplies all the variables defined but not used. Program errors that result from incorrect names (e.g. IO instead of IO) are very difficult to find.

5. All the parameters that are important for program control should be checked for their validity.

   REASON - EXPLANATION

   In this way a higher degree of stability in the running of the program is achieved. It is especially important, for example, to check the user inputs, since first of all the user often loses interest when a program frequently breaks off, and, secondly, no unnecessary computer time is wasted if, for example, the program runs in an endless loop. /9

6. COMMON blocks should be avoided. If a COMMON block is still used

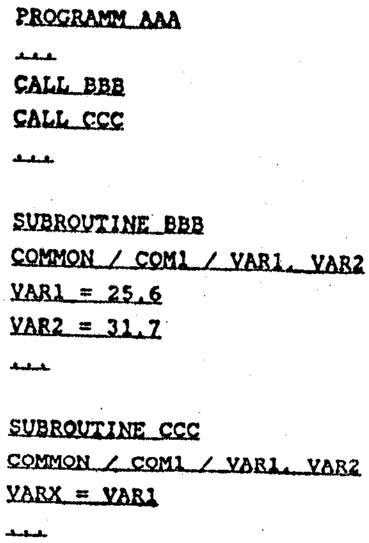
this should be inserted into the actual source with "INCLUDE".

REASON - EXPLANATION

COMMON blocks are highly subject to error, e.g. as a result of a typing error (incorrect variable name) or an incomplete change (1 COMMON block in 20 sub-programs, but only changed in 19) errors may occur that are not found by a compiler and that a programmer may find only after searching for days using test results.
In addition, the realization of the COMMON block depends on the compiler, which in the case of program re-use may produce quite different results.
The COMMON block is permitted for global variables that are not changed in the program, for example, for variables that indicate the logical data numbers (FT numbers). If a COMMON block is used, then the probability of error can be significantly reduced by means of the INCLUDE command. The drawback is that the INCLUDE instruction is compiler-dependent.

Example of a typical COMMON block error.

/10



```
PROGRAMM AAA
...
CALL BBB
CALL CCC
...

SUBROUTINE BBB
COMMON / COM1 / VAR1, VAR2
VAR1 = 25.6
VAR2 = 31.7
...

SUBROUTINE CCC
COMMON / COM1 / VAR1, VAR2
VARX = VAR1
...
```

When BBB is first called up, the COMMON block is occupied; when BBB is dropped, the COMMON is undefined, since it is not given in the main programm AAA. Thus the COMMON block in CCC ia also undefined. VARX receives an indefinite value.

1.2 Rules for setting up sub-programs

1. A sub-program (subroutine, function) should not exceed a length of 100 (approx.) FORTRAN instructions that are to be carried out.

   REASON - EXPLANATION

   Experience has shown that sub-programs contain too many functions and as a result they become unwieldy and difficult to change. The smaller the program unit, the more likely are the test possibilities.

2. The argument list (parameter list) of a sub-program should be /11
   arranged as follows: input, input/output, output, auxiliary variables/fields, error code.

   REASON - EXPLANATION

   The rule makes it easier to oversee the programs and facilitates documentation (indication of the sequence in the parameter list).

3. At the end of every parameter list of any sub-program there is an error code that is inserted in the sub-program and from which enquiries can be made by the program being called up. This error code should be of the INTEGER type. The value 0 (zero) defines NO ERROR; every other value designates an error. The arrangement error <-> value should always be unambiguous in a program.

   REASON - EXPLANATION

   In this way a definite central error processing is possible that can implement either program continuation or a controlled program interruption. For example, important partial results can still be stored or control variables can be returned to starting values.

In addition, in this way the "RETURN-i" instruction can be omitted.

4. In a "CALL" instruction, the following parameters should not be given as arguments:

   - Constants, e.g. CALL SUB (2.5)
     Exception: text constants
   - Arithmetical expressions, e.g. CALL SUB (4+5*A)
   - Function summons, e.g. CALL SUB (FUNK(X))

   REASON - EXPLANATION

   /12 If a variable that was given as a constant is changed in a program that has been called up, then this may produce errors in the program called up that are difficult to pinpoint.

Example: If the subroutine TEST is called up as follows

```
...
CALL TEST (10)
...
J = 10
```

and the parameter in TEST is changed as follows

```
SUBROUTINE TEST (I)
...
I = 20
...
```

then the variable J (in most compilers) has the value 20, since then only the address of the memory cell in which the 10 is located is given and this is carried over in the sub-program.

1.3 Rules for arithmetical expressions

1. In the case of arithmetical expressions in floating point computation, the reference operands ".EQ." and ".NE." are forbidden.

REASON - EXPLANATION

As a rule, floating point numbers are not stored exactly in a computer. Since, in addition, when arithmetical expressions are evaluated rounding errors almost always occur, the probability that a result contains the anticipated value exactly is equal to zero.

2. Expressions that contain operands of different types are not permissible.

REASON - EXPLANATION /13

If the conversion routines are not used, then the result depends on the computer. Example:

```
INTEGER N
N = 2 * 3.7
```

==> N = 6 if 3.7 is converted to 3 before the calculation.

==> N = 7 if 7.4 is converted to 7 after the calculation.

3. Parentheses should always be used if the result (numerical value) of an arithmetical expression depends on whether the calculation is carried out from left to right or right to left.

REASON - EXPLANATION

The same problem arises here also, Example:

```
INTEGER N
N = 10 * 3 / 5
```

==> N = 6 if 10*3=30 and 30/5=6 is calculated.

==> N = 0 if 3/5=0.6, INT(0.6)=0, then 10*0=0.

1.4 Use of CHARACTER parameters

1. A CHARACTER parameter with more than 255 signs is forbidden.

REASON - EXPLANATION

A greater length is compiler-dependent.

2. It is not permitted to compare CHARACTER parameters with the reference operators for numbers (e.g. ".LT.").

   REASON - EXPLANATION

   The result of comparison operations is not defined, and thus /14 it is computer-dependent. While the intrinsic functions "LGE", "LGT", "LLE" and "LLT" for CHARACTER deliver as a rule an established result.

3. The FORTRAN functions "ICHAR" and "CHAR" are not permitted.

   REASON - EXPLANATION

   These two functions supply a system-specific value; therefore they are unsuitable for the INTEGER -> CHARACTER conversion (and vice versa) . If a conversion of this type is unavoidable, then this itself should be programmed.

4. The conversion of a CHARACTER field into a CHARACTER variable by calling up a sub-program is forbidden.

   REASON -EXPLANATION

   In this way a conversion is carried out that as a rule is not defined and therefore may produce varying results.

5. All the CHARACTER parameters given via the argument list in a sub-program should be variably dimensioned ("CHARACTER*(*)").

   REASON-EXPLANATION

   In this way a variable CHARACTER length can be given , which considerably facilitates programming and avoids errors. Unfortunately, this makes it impossible to carry out a field limits check, which some new compilers can do.

6. If a CHARACTER parameter in a sub-program is dimensioned with "*", then it no longer sensible to give the actual length via the argument list.

REASON - EXPLANATION

Using the "LEN" function, it is possible to enquire the actual length /1 of the symbol chain or the actual length can be given out using the FORMAT datum A without knowing the length. In this way an additional parameter is avoided in the argument list.

1.5 Programming of the DO instruction

1. After the DO "label" there should be a comma (e.g. DO 100, I = 1, N).

   REASON - EXPLANATION

   Here is an illustrative example from history:

   `DO 3 I=1.3` instead of `DO 3 I=1.3`

   In the program for the flight path calculation of the first Venus probe. Mariner 1, instead of the comma in the DO loop, a period was written. As a result the variable "DO3I" was made equal to 1.3. This meant that the space probe had to be exploded after 290 seconds on July 22, 1962.

2. Each DO instruction should be ended with the label "CONTINUE".

   REASON - EXPLANATION

   Another example:

```
      N = 0
      DO 100, I = 1, 10
        J = I
        DO 100, K = 5, 1
          L = K
  100 N = N + 1
```

   What is the value of N after the program has been carried out?

   As a rule this example is interpreted as follows:

/16

```
    N = 0
    DO 100, I = 1, 10
      J = I
      DO 200, K = 5, 1
        L = K
        N = N + 1
200   CONTINUE
100 CONTINUE
```

I.e. N receives the value 0 (I=11, J=10 and L has no value), since the inner DO loop is not carried out.

If its own instruction number (LABEL) is assigned to the inner DO block, then it is much easier at any time to insert an instruction after the inner DO loop and before the end of the outer DO loop.

3. Loop variables of the REAL or DOUBLE PRECISION type should not be used if possible.

REASON - EXPLANATION

As a result of rounding errors, the actual number of loop runs can be varied or automatically corrected via the compiler.

1.6 Rules for assigning instruction numbers

1. The instruction numbers may only be used in conjunction with a CONTINUE or FORMAT instruction.

REASON - EXPLANATION

By means of CONTINUE it is clearly defined which command is carried out before the sentinel and which is carried out after it; a typical error has already been shown in the DO instructions.

2. The sequence of the numbers should be given in ascending order; the intervals should be as large as possible (100 times increments). /17

REASON - EXPLANATION

This considerably facilitates the sequential search in a program and the insertion of new, additional numbers.

3. For the FORMAT statements, at the end of the program its own number range (e.g. from 9000 onwards) should be given (between STOP/RETURN and END).

   REASON - EXPLANATION

   In this way it is easier to monitor a program, since, first of all, the FORMAT instruction numbers are not valid sentinels for jump errors, and, secondly, FORMAT instructions are not detailed instructions; thirdly, they are often very long, which in the case of inserted structures would result in too many continuation statements.

4. The last CONTINUE instruction directly before the STOP or RETURN instruction should receive throughout the program a uniform, easily recognizable high number (e.g. 8888 or 9999).

   REASON - EXPLANATION

   In this way jumps to the end of the program (e.g. in the event of error) can be immediately recognized.

1.7 Forbidden FORTRAN-77 instructions

1. The assigned GOTO instruction.

   REASON - EXPLANATION

   The assigned GOTO instruction contradicts the rules of structured programming. Programs that contain this statement are almost impossible to maintain, even for the author of the program.

2. The arithmetical IF instruction

   REASON - EXPLANATION

   The same applies to tne arithmetical IF, although it is not quite so important, since by means of GOTO instructions a jump can be made to a defined block end. In this way, again, far too

/18

many jumps occur (GOTOs), which makes the program subject to errors and unmanageable.

3. The computed GOTO instruction, with restrictions

REASON - EXPLANATION

The computed GOTO instruction is only permitted in order to realize the CASE structure; after the GOTO statement an error block should follow and a "GOTO->CASE end" should stand in front of every CONTINUE statement. In all other cases, this instruction is highly subject to error.

4. The OPEN, CLOSE and INQUIRE instructions may only occur in special sub-programs for data processing.

REASON - EXPLANATION

Normally these instructions are not clearly defined. Therefore various manufacturers offer different forms of implementation (e.g. IBM <-> Fujitsu, ...). If these instructions cannot be omitted, then they should be isolated in code.

5. The PAUSE instruction

REASON - EXPLANATION

The PAUSE instruction can be readily replaced by a WRITE with subsequent READ and thus becomes computer-independent.

6. The EQUIVALENCE instruction for variables of various types

REASON - EXPLANATION /19

If the EQUIVALENCE instruction is used for variables of different types, this is impermissible trick programming.

7. The BLOCK DATA instruction

REASON - EXPLANATION

Both the assignment of the BLOCK DATA instruction in the program code and the storage site assignment in the program design vary with the different manufacturers. For this reason, as a result of the BLOCK DATA instruction errors occur that are often difficult to find.

8. All instructions that are not normally received

REASON - EXPLANATION

Onecan find out what is the norm in ANSI FORTRAN X3.9-1978 (Fortran 77) or in ISO 1539-1980 Programming Language Fortran Manual (both standards are technically the same as to content).

1.8 Directions for the input and output functions

1. Input and output functions should be carried out by means of the relevent sub-programs, in agreement with the EVA rule.

2. Variables should be used for logical data numbers that may be employed at one central point.

REASON - EXPLANATION

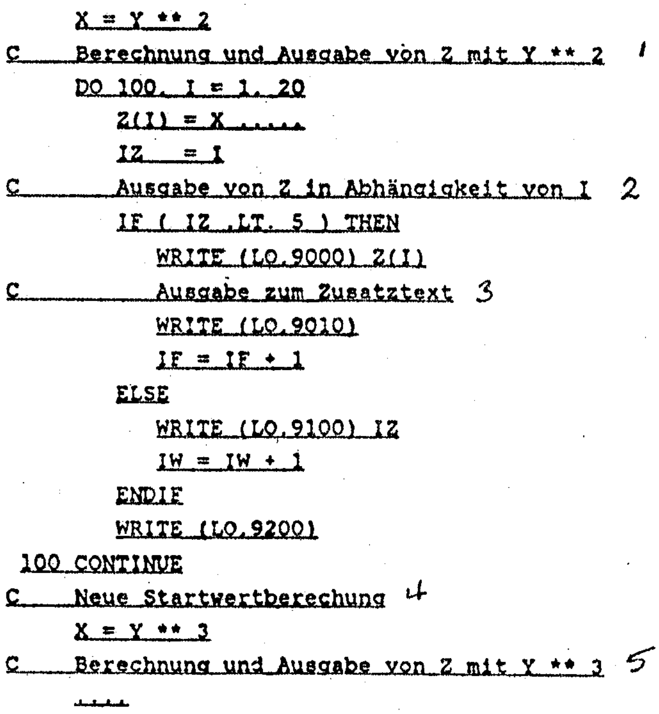
The input/output regulation is especially affected by computer-dependence. If these functions are only used or engaged at one point in the program, then the work required to make changes is much less if a conversion is made to another computer.

1.9 Code encapsulation /20

In accordance with the blocks in the structured programming, the statements have to be put into code (2-3 columns per capsule depth). This means at the same time that only one statement may be written per line.

REASON - EXPLANATION

This precaution decidedly improves the manageability of a program. In the appendix on p. 23 is shown how the blocks in structured programming should be converted to FORTRAN 77. Here is a short example, as well[2]:



```
          X = Y ** 2
    C     Berechnung und Ausgabe von Z mit Y ** 2  1
          DO 100, I = 1, 20
             Z(I) = X .....
             IZ   = I
    C        Ausgabe von Z in Abhängigkeit von I  2
             IF ( IZ .LT. 5 ) THEN
                WRITE (LO,9000) Z(I)
    C           Ausgabe zum Zusatztext  3
                WRITE (LO,9010)
                IF = IF + 1
             ELSE
                WRITE (LO,9100) IZ
                IW = IW + 1
             ENDIF
             WRITE (LO,9200)
     100 CONTINUE
    C     Neue Startwertberechung  4
          X = Y ** 3
    C     Berechnung und Ausgabe von Z mit Y ** 3  5
          ....
```

Key: 1 calculation and output of Z with Y ** 2;
2 output of Z in relation to I;
3 output to supplementary text;
4 new starting value calculation;
5 calculation and output of Z with Y ** 3.

1.10 Functions dependent on software and hardware /21

Parts of the program that depend on the operational system, the computer and the compiler, as well as library sub-programs, should only be used in their own modules and they should be clearly identified there.

[2] This example does not make any claim to be scientific or meaningful, but it should make clear the advantage of the insertions.

REASON - EXPLANATION

Among these are:

maximum CHARACTER length;
maximum memory space;
maximum number of dimensions
- of continuation statements,
- of sub-program parameters,
- of encapsulation of IF and DO loop blocks,
- of "CASE" function blocks,
- of DATA instructions;

statement length for I/O;
field length;
largest and smallest number that can be presented;
mechanical accuracy;
screen control (dimming, masking, etc.);
graphics programs;
mathematical library sub-programs;
datum, CPU time and clock time sub-programs.

and many more.

2. Rules for program documentation

Good program documentation, which is prepared during programming, is just as important as all the coding guidelines put together in order to prepare user-friendly, reliable and portable programs. Detailed documentation should not be limited to code documentation, but should also include the entire programming environment. Since good, detailed and specialized documentation is important in general, no reasons are given for the following rules. /22

1. A description at the head of every program or sub-program should explain its function and the interfaces. In addition, information should be given about the preparer, the caretaker, the status (date) and peculiarities of the program (see also p.30).

2. In the program code, a commentary should explain the following instructions in technical terms before

   the start of every iteration or loop,
   a sequence of logically associated instructions,
   an IF instruction or an IF-THEN-ELSE structure,
   a CALL instruction,
   a (non-avoidable) GOTO instruction.

3. The code is even easier to read if there are empty lines between the individual blocks as well as by means of the already mentioned code encapsulation and by means of the deliberate use of blanks. For example:

```
PI     = 3.14                       PI=3.14
RXMAX  = 10.       instead of       RMAX =10.
EULERZ = 2.7182                     EULERZ = 2.7182
G      = 9.81                       G= 9.81
```

4. Each later change in the program should be noted in the head commentary, giving the reason, reference, date and name, and the section changed, with reason and abbreviation.

5. In extensive programs a description of the general structure of the entire program (i.e. division into modules) should be included in the main program or in a kind of appendix (realized /23 by means of INCLUDE). A too detailed description at this point (fine structure) is unrealistic, since this would have to be changed far too often and a documentation that is not a true one is useless and dangerous.

6. With the aid of name conventions for sub-programs and variables it is also possible to achieve a code documentation. For example, the program structure can be simulated in the sub-program names, Nxxxxx for numerical sub-programs, Exxxxx for input sub-programs, Xxxxxx for computer-dependent sub-programs, etc.

3. Summary

In principle all the rules explained here should be used for any kind of programming. Unfortunately, this will not always be possible; for example, in real time programming. These rules are absolutely essential for the development of long-life, portable program systems. However, who knows in the initial stages of a program development how long this program will live. Therefore, these rules should always be observed and any trick programming should be eliminated.

The directions given here can be weighted according to the following aspects:

1. General rules that help to avoid many programming errors.
2. Rules for improving the manageability of a program.
3. Rules that enhance the reliability of a program.
4. Rules for portable programs.

The weighting of these rules should be done by each program developer himself or herself (for an example, see p. 31).

/24 These guidelines were developed at ESA (European Space agency) for FORTRAN 66 in the 1970s already. For this version the ESA rules were supplemented with new information for FORTRAN 77 by the working group Software Engineering (abbreviation ASE) of the DFVLR.

Considerable contributions to these guidelines were made by Mr. Martin Otter, DFVLR - FF-DF Oberpfaffenhofen, and Mr. Wilfried Sagrauske, DFVLR - WT-DV Brunswick.

The ASE would be indebted for any suggestions for improving these guidelines.

4. References /25

GEWALD, K.; HAAKE, G.; PFADLER, W.
Software Enginee.ing:
Grundlagen und Techniken moderner Programmentwicklung.
R. Oldenbourg Verlag, 3.Auflage 1982.

KURBEL, KARL
Programmierstil
in Pascal, Cobol, Fortran, Basic, PL/I.
Berlin, Heidelberg, Springer Verlag, 1985.

SNEED, HARRY M.
Software Entwurfsmethodik.
SOFTORG-Seminar, 1985.

METCALF, MICHAEL
FORTRAN - Optimization.
ACADEMIC PRESS, 1982.

COMPAS' PROCEEDINGS 85
Computer Anwendungen, Software und Systeme.
Berlin, Offenbach, VDE-Verlag Gmbh, 1985.

AMERICAN NATIONAL STANDARD PROGRAMMING LANGUAGE FORTRAN
ANSI X3.9 - 1978
New York

OTTER, MARTIN
FORTRAN 77 - Richtlinien
für RASP und ANDECS.
DFVLR FF-DF, 1986.

SCHUSTER, WOLFGANG
Strukturierte Programmierung und Codierung in Fortran 66.
DFVLR FF-DF, 1981.
WT-DA Software Engineering Standards.
Programmieren in FORTRAN 77.
Hauptabteilung Angewandte Datentechnik,
DFVLR, Oberpfaffenhofen, 1986. /26

Key to references:

Gewald, K., Haake, G., Pfadler, W.:
Software Engineering: Fundamentals and Techniques of Modern Program Development.
R. Oldenbourg Verlag, 3rd.ed., 1982.

Kurbel, Karl:
Programming Style in Pascal, Cobol, Fortran, Basic, PL/I.
Berlin, Heidelberg, Springer Verlag, 1985.

Sneed, Harry M.:
Software Development Methods. SOFTORG Seminar, 1985.

Compas' Prceedings, 1985:
Computer applications, software and systems.
Berlin , Offenbach, VDE Verlag GmbH, 1985.

Otter, Martin:
FORTRAN 77 - Guidelines for RASP and ANDECS.
DFVLR FF-DF, 1986.

Schuster, Wolfgang:
Structured programming and coding in Fortran 66.
DFVLR FF-DF, 1981.

WT-DA Software Engineering Standards.
Programming in FORTRAN 77.
Main division, Applied Data Technology,
DFVLR, Oberpfaffenhofen, 1986.

Structure blocks according to Nassi-Shneiderman and their conversion into Fortran 77 /27

SB : = structure block

la : = label/sentinel - start of structure block

le : = label/sentinel - end of structure block

ll : = label/sentinel - continuous from 1 to n

lx : = label/sentinel

is : = loop index

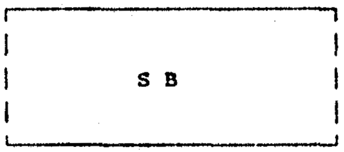
ia : start of loop

ie : = end of loop

C==== : = start of structure block - 1st. degree

C---- : = start of structure block - 2nd. degree
(only for clarification of the description)

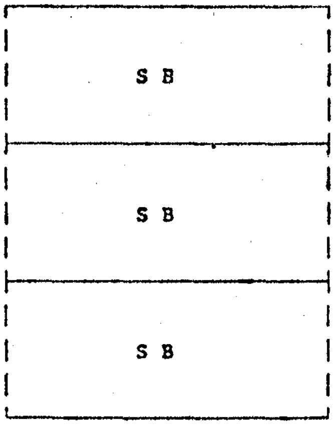
Elementary block



```
C====
C     Anweisungsblock
      .....
      .....
C
```

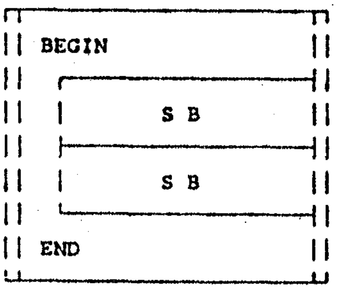
Key: Anweisungsblock = instruction block

Sequence



```
C====
C     Anweisungsblock 1
          .....
          .....
C----
C     Anweisungsblock 1
          .....
          .....
C----
C·    Anweisungsblock 1
          .....
          .....
C
```

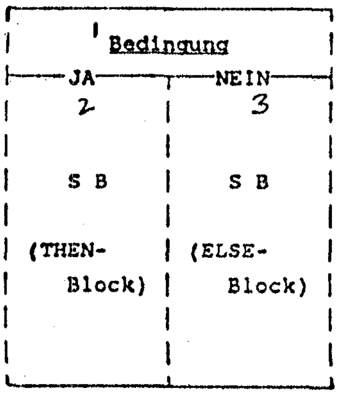
Procedure



```
C====
C     Prozedur 2
      PROGRAM / SUBROUTINE <Name>
C----    Anweisungsblock 1
          .....
C----    Anweisungsblock 1
          .....
C
      END
C
```

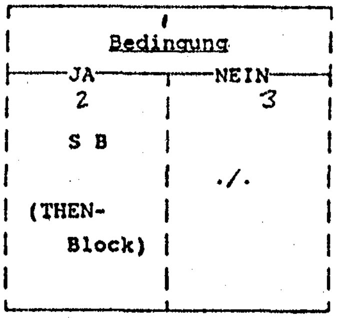
Key: 1 instruction block; 2 procedure.

Conditional branching or alternative instruction



```
C====               1
        IF <Bedingung> THEN
C
C----       THEN-Anweisungsblock
            .....                  4
            .....
        ELSE
C----       ELSE-Anweisungsblock
            .....              4
            .....
        ENDIF
C
```

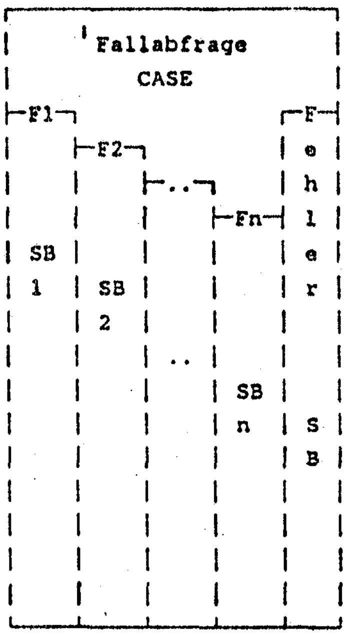
Conditional branching without ELSE block



```
C====               1
        IF <Bedingung> THEN
C
C----       THEN-Anweisungsblock
            .....                  4
            .....
            .....
            .....
        ENDIF
C
```

Key: 1 condition; 2 yes; 3 no; 4 instruction block.

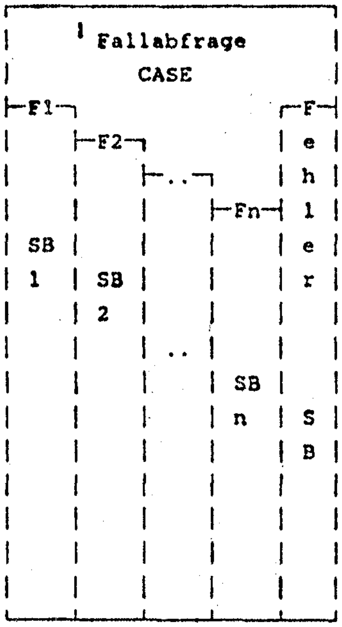
Case differentiation with error output (general)



```
                     2
C====   IF (<Ausdruck F1>) THEN
C----      F1-Block
           .....
           .....
           .....
                            2
        ELSE IF (<Ausdruck F2>) THEN
C----      F2-Block
           .....
           .....
        ELSE IF (....
           .....
           .....
           .....
        ELSE
C----      Fehlerblock 3
           .....
C          Ende - Fehlerblock 4
C          und CASE-Struktur
        ENDIF
```

Key: 1 case enquiry; 2 expression; 3 error block; 4 end of error block and CASE structure.

Special case for an expression of INTEGER type

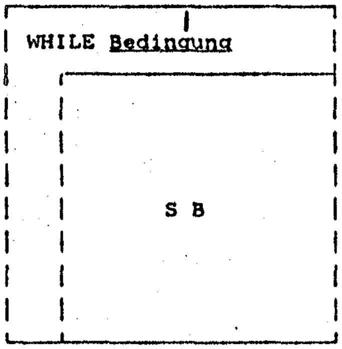


```
C====                                2
       GO TO (11,12,....), <Ausdruck>
C----     Fehlerblock 3
          .....
          .....
          GO TO le
 11    CONTINUE
C----     F1-Block
          .....
          .....
          GO TO le
 12    CONTINUE
C----     F2-Block
          .....
          .....

          .....
 le    CONTINUE
```

Key: 1 case enquiry; 2 expression; 3 error block.

Repetition, head-controlled loop

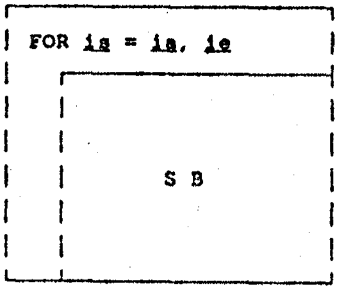


```
C====
 la    CONTINUE
                  1
       IF (.NOT.<Bedingung>) GO TO le
C----     Anweisungsblock 2
          .....
          .....
          .....
       GO TO la
 le    CONTINUE
```

Key: 1 condition; 2 instruction block.

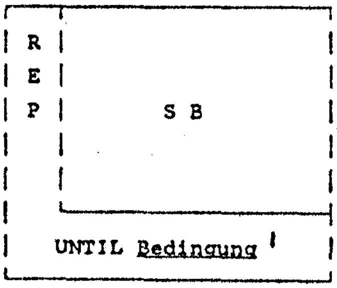
/32

Special case



```
C====
      DO lx, is= ia, ie
C----    Anweisungsblock 2
         .....
         .....
         .....
         .....
  lx  CONTINUE
```

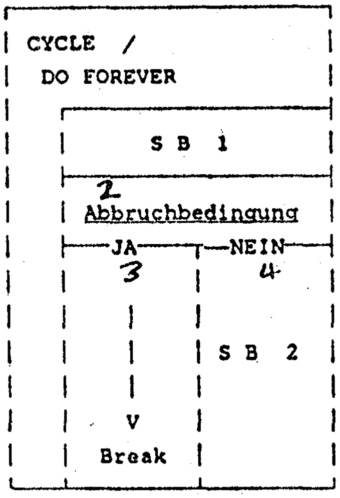
Repetition, foot-controlled loop



```
C====
  lx  CONTINUE
C----    Anweisungsblock 2
         .....
         .....
         .....
         .....
      IF (.NOT.<Bedingung>) GO TO lx
                    1
```

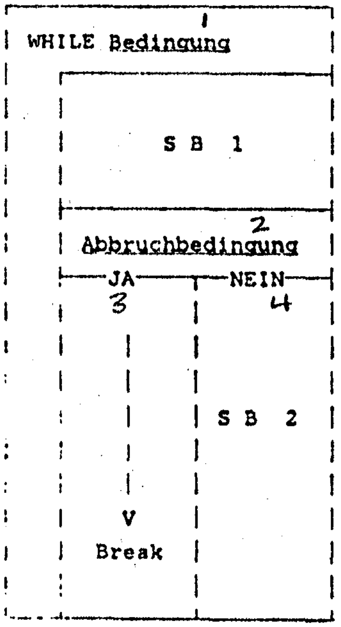
Key: 1 condition; 2 instruction block.

Repetition, loop with interruption /33



```
C====
 la   CONTINUE
C----    Anweisungsblock 1
                  5
         .....
         .....                        6
C        Erklärung d. Abbruchbed.
         IF (<Abbruchbed.>) GOTO LE
                     2
C
C----       Anweisungsblock 2
                  5
            .....
            .....
C
      GO TO la
 le   CONTINUE
```

Repetition, head-controlled loop with interruption /34



```
C====
 la   CONTINUE
                  1
      IF (.NOT.<Bedingung>) GO TO le
C
C----    Anweisungsblock 1
                  5
         ......
         ......
         ......
C
                   2
         IF (<Abbruchbed.>) GO TO le
C
C----       Anweisungsblock 2
                  5
            .....
            .....
C
      GO TO la
 le   CONTINUE
```

Key: 1 condition; 2 interrupt condition; 3 yes; 4 no; 5 instruction block; 6 statement of interrupt condition.

Model for program commentary head

[Key on foll. page]

```
CA=============================...=============================
C   1 SUBROUTINE <Pgm.-Name>    2 STAND               : tt.mm.jj
C                               3 LETZTER BEARBEITER: <Kürzel> 4
C==============================...=============================
C     5                           6
C FUNKTION: Kurzbeschreibung der Funktion des Unterprogrammes
C       7                       8
C AUFRUF: CALL <PGM.-Name> (<Parameterliste>)
C          9
C BESCHREIBUNG DER PARAMETER:
C  Name, Art (Eingabe/Ausgabe/Hilfsfeld/..), Typ (Real, ... 10
C  Dimension und Kurzbeschreibung 11
C
C BESCHREIBUNG DER COMMON-BLOECKE: 12
C  Verwendung und Parameterbeschreibung wie oben  13
C      14
C BEMERKUNGEN: Hinweis auf Besonderheiten 15
C        16
C LITERATURVERWEISE: falls vorhanden 17
C
C==============================...=============================
C        18
C PROGRAMMIERSPRACHE    <Fortran 66/77> / <PL/I>, ...
C AUFRUFSPRACHE 19      - " -
C BETRIEBSSYSTEM 20     IBM - MVS/VM, VAX / VMS/RSX, ...
C BETRIEBSART 21        Dialog und/oder Batch  22
C GERAETE 23            benoetigte externe Geräte  24
C BENOETIGTE UPS        Liste der aufgerufenen Unterprogramme  26
C              25       / Bibliotheken
C COMMON                Namensliste 27
C AUTOR 28              Name und Zuordnung  29
C BETREUER 30           - " -
C BESCHREIBUNG 31       Verweis zur Dokumentation  32
C
CE=============================...=============================
```

Key to model:

1 sub-routine; 2 status; 3 last user; 4 abbrev.; 5 function; 6 brief description of function of sub-program; 7 call-up; 8 parameter list; 9 description of parameters; 10 name, type (input/output/auxiliary field/...), type (actual, ...); 11 dimension and brief description; 12 description of common blocks; 13 application and description of parameters as above; 14 remarks; 15 indication of peculiarities; 16 references; 17 if available; 18 program language; 19 call-up language; 20 operating system; 21 form of operation; 22 dialog and/or batch; 23 equipment; 24 external equipment required; 25 sub-programs required; 26 list of sub-programs called up / libraries; 27 list of names; 28 author; 29 name and assignment; 30 caretaker; 31 description; 32 indication of documentation.

Simplified proposal according to the draft worked out by the Working Group on Numeration at the Computer Center (IB 81-162/05). /36

Weighting of the rules

In the following table a weighting of the rules is attempted in terms of the four aspects/ weighting criteria mentioned in the text.

The number of asterisks indicates the degree of importance.

1 Wichtungskriterien	2 Fehlervermeidung	3 Wartbarkeit	4 Zuverläßig keit	5 Portabilität
6 Regeln				
1.1 7 Allgemeine Regeln				
1.1.1 8 Modularität	***	*****	***	****
1.1.2 9 nur ein Ein- und Ausgang	**	***	*	*
1.1.3 10 Programmfluß vorwärts	***	***	*	
1.1.4 11 Variablendeklaration	***	**	*	**
1.1.5 12 Prüfung der Steuervariablen	***	*	***	
1.1.6 13 COMMON Blöcke	*	***	*	***
1.2 14 Unterprogramme				
1.2.1 15 Länge 100	*	***		
1.2.2 16 Parameterliste E,E/A,A,H,F	*	**		
1.2.3 17 Fehlercode	**	***	***	
1.2.4 18 ungeeignete Übergabeparameter	**	*	*	***

Key: 1 weighting citeria; 2 error avoidance; 3 manageability; 4 reliability; 5 portability; 6 rules; 7 general rules; 8 modularity; 9 only one input and output; 10 program flow forwards; 11 end declaration of variable; 12 testing of control variables; 13 COMMON blocks; 14 sub-programs; 15 length; 16 parameter list; 17 error code; 18 unsuitable transmission parameters.

	1	2	3	4	5
	Wichtungskriterien	Fehlervermeidung	Wartbarkeit	Zuverläßigkeit	Portabilität
6	Regeln				
7	1.3 arithmetische Ausdrücke				
8	1.3.1 illegale Vergleichsoperatoren	***	*		***
9	1.3.2 verschiedene Typen	***	*	*	***
10	1.2.3 Klammerung	**	*		**
11	1.4 CHARACTER-Größen				
12	1.4.1 Länge 255	*	***		****
8	1.4.2 illegale Vergleichsoperatoren	*	*	*	**
13	1.4.3 ICHAR und CHAR	*		**	***
14	1.4.4 Feld <-> Variable	*		*	**
	1.4.5 *(*) Dimensionierung 15	**	**		*
16	1.4.6 Länge als Parameter bei *	*	**		

Key: 1 weighting criteria; 2 error avoidance; 3 manageability; 4 reliability; 5 portability; 6 rules; 7 arithmetical expressions; 8 illegal comparison operators; 9 different types; 10 parentheses; 11 CHARACTER sizes; 12 length; 13 ICHAR and CHAR; 14 field <-> variable; 15 dimensioning; 16 length as parameter for *.

	1	2	3	4	5
	Wichtungskriterien	Fehlervermeidung	Wartbarkeit	Zuverläßig keit	Portabilität
6	Regeln				
7	1.5 DO-Anweisung				
8	1.5.1 Komma nach "DO label"	***			
	1.5.2 CONTINUE	**	***		
9	1.5.3 Schleifenvariable ¬= REAL, ..	**	*	**	***
10	1.6 Anweisungsnummern				
11	1.6.1 nur CONTINUE oder FORMAT	***	***	*	
12	1.6.2 Reihenfolge	*	***		
13	1.6.3 eigener FORMAT-Nummernbereich	*	***		
14	1.6.4 CONTINUE am Programmende	*	**		

Key: 1 weighting criteria; 2 error avoidance; 3 manageability; 4 reliability; 5 portability; 6 rules; 7 DO instruction; 8 comma after "DO label"; 9 loop variable; 10 instruction numbers; 11 only CONTINUE or FORMAT; 12 sequence; 13 own FORMAT numbers zone; 14 CONTINUE at end of program.

1 Wichtungskriterien 6 Regeln	2 Fehlervermeidung	3 Wartbarkeit	4 Zuverläßigkeit	5 Portabilität
1.7 7 Verbotene Anweisungen				
1.7.1 assign GOTO	****	****	***	
1.7.2 8 arithmetische IF	***	****	**	
1.7.3 computed GOTO	**	***	*	
1.7.4 9 OPEN, CLOSE und INQUIRE		*		***
1.7.5 PAUSE	*	**		***
1.7.6 EQUIVALENCE	**	**	*	**
1.7.7 BLOCK DATA	**	**		**
1.7.8 Norm	**	**	*	****
1.8 10 Ein-/Ausgabefunktionen	*	***		****
1.9 11 Codeschachtelung	*	****		
1.10 9 Soft- und Hardware	*	***		****

Key: 1 weighting criteria; 2 error avoidance; 3 manageability; 4 reliability; 5 portability; 6 rules; 7 forbidden instructions; 8 arithmetical IF; 9 and; 10 input/output functions; 11 code encapsulation.

1	2	3	4	5
Wichtungs-kriterien	Fehler-ver-meidung	Wart-bar keit	Zuver-läßig keit	Porta-bilität
6 Regeln				
2. 7 Dokumentation				
2.1 8 Kopfkommentar	**	****		*
2.2 9 Code-Kommentierung	**	****		*
2.3 Code-Design	*	***		
2.4 10 Änderungskommentierung	***	****		*
2.5 11 Strukturbeschreibung	*	***		*
2.6 12 Namenskonventionen	**	***		

Key: 1 weighting criteria; 2 error avoidance; 3 manageability; 4 reliability; 5 portability ; 6 rules; 7 documentation; 8 head commentary; 9 code commentary; 10 change commentary; 11 structure description; 12 name conventions.

Research field: Flight mechanics/flight guidance /42

Main office: Flughafen, D-3300 Brunswick

Institute for Flight Mechanics

Institute for Flight Guidance

Institute for Dynamics of Flight Systems

Institute for Flight Medicine

Research field: Flow mechanics

Main Office: Bunsenstrasses 10, D-3400 Goettingen

Institute for Theoretical Flow Mechanics

Institute for Experimental Flow Mechanics

Institute for Drive Technology

Institute for Aerodynamic Design

Research field: Materials and methods of construction

Main Office: Pfaffenwaldring 38-40, D-7000 Stuttgart

Institute for Structural Mechanics

Institute for Aero-elasticity

Institute for Materials Research

Institute for Space Simulation

Institute for Construction Methods and Research

Research field: Communications technology and reconnaissance

Main Office: D-8031 Oberpfaffenhofen, Post Wessling/Obb.

Institute for High Frequency Technology

Institute for Opto-electronics

Institute for Atmospheric Physics

Institute for Communications Technology

Research field: Energetics

Main Office: Pfaffenwaldring 38-40, D-7000 Stuttgart

Institute for Technical Physics

Institute for Technical Thermodynamics

Institute for Physical Chemistry of Combustion

Institute for Chemical Impulses and Methods Technology

Field: Space travel
Main Office: Cologne-Porz/Oberpfaffenhofen

Field: Scientific/technical operating systems
Main Office: D-8031 Oberpfaffenhofen, Post Wessling/Obb.

Main division: Wind channels
Main Office: Bunsenstrasse 10, D-3400 Goettingen

Main division, Traffic research
Linder Hoehe, D-5000 Cologne 90 (Porz)

Project Support: Labor, Environment, Health
Main Office: Suedstrasse 125, D-5300 Bonn 2

Publications of the DFVLR /43

DFVLR Research Reports
Scientific first publications from the research and development work of the DFLVR

DFLVR Reports
Contributions on test methods and systems, computer programs, lectures, reference lists for certain subjects

Annual index
of DFLVR research reports and DFLVR reports (German and English)

Zeitschrift fuer Flugwissenschaften und Weltraumforschung (ZFW)
Jounral with contributions on actual research and development studies, with indications of DFLVR publications, DFLVR inventions and patents, project support reports and events

Scientific reports in various fields /43

Overview of the most important research and development projects, structure, spectrum of assignments in these fields, and prospects of future development (German and English)

Inventions and patents

List of patents applied for and those awarded

Report Documentation Page

1. Report No.
TT-20317

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle
DIRECTIONS FOR PROGRAMMING IN FORTRAN 77

5. Report Date
JULY 1988

6. Performing Organization Code

7. Author(s)
Hildegard Schuster

8. Performing Organization Report No.

9. Performing Organization Name and Address
National Aeronautics and Space Administration
Washington, DC 20546

10. Work Unit No.

11. Contract or Grant No.
NASW-4307

12. Sponsoring Agency Name and Address

13. Type of Report and Period Covered
translation

14. Sponsoring Agency Code

15. Supplementary Notes
Transl. from GERMAN to ENGLISH of "RICHTLINIEN ZUR PROGRAMMIERUNG IN FORTRAN 77". DFVLR-MITT-87-10, 1987,pp. 1-41

Transl. by SCITRAN, Santa Barbara, CA 93150.

16. Abstract
With the introduction of programming directions it is possible to reduce long-term costs of software development and maintenance. The foundation of every programming should be based on the rules of the structured programming. For FORTRAN 77 programming these rules are insufficient, therefor further rules must be defined. Some of these programming rules can be simply used for other programming languages.

17. Key Words (Suggested by Author(s))

18. Distribution Statement
unclassified-unlimited

19. Security Classif. (of this report) unclassified	20. Security Classif. (of this page) unclassified	21. No. of pages 42	22. Price

N88-15487#(o
N-157,477
N88-26842#